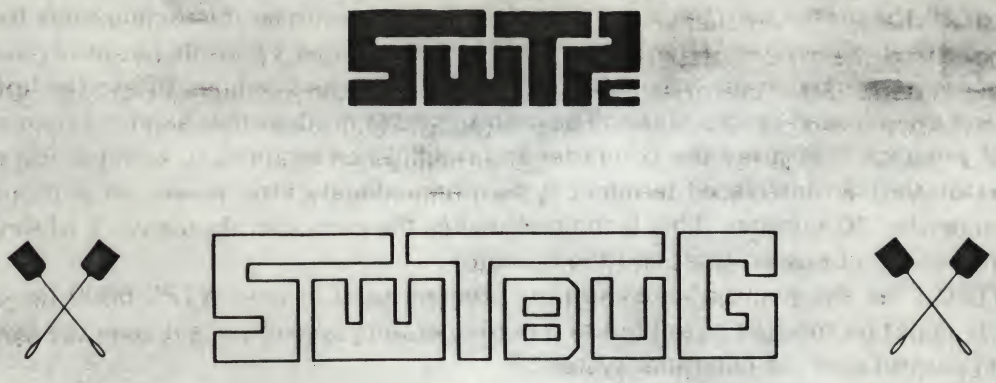
ACIAIN E440
ACIOUT E44C



6800 ROM MONITOR

VERSION 1.0

USERS GUIDE

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. Rhapsody San Antonio, Texas 78216

SWTPC SWTBUG® (SWATBUG) MONITOR ROM

One of the features of the SWTPC 6800 Computer System is that the conventional programmer's console has been replaced with a monitor ROM. The programmer's console consists of all the pretty switches and lights often found on similar microcomputers that are used to bootstrap the system after power up. The programmer's console not only raises the cost of the system, but more often than not is confusing and tedious to use for both beginning and experienced programmers. The monitor ROM on the other hand is a permanently stored program that gives the computer the intelligence required to communicate with the operator thru an interfaced terminal system immediately after power up without flipping switches for 10 minutes. This technique makes the computer do the work of simplifying communication between itself and the operator.

SWTBUG® is the name of the monitor program used in the SWTPC 6800 Computer System. It might be thought of as kind of a mini-operating system since it gives the operator command control over the computer system.

Features of the SWTBUG®ROM include:

* Memory Examine and Change
* Program loading from cassette or paper tape thru the control interface or thru I/O port # 0.
* Program saving to cassette or paper tape
* Go to user program
* Display contents of registers
* Erase SWTPC CT-1024 terminal system screen
* SWTPC MF-68 floppy disk boot
* Byte search
* Breakpoint debugging
* Vectored hardware and software interrupts to user defined addresses

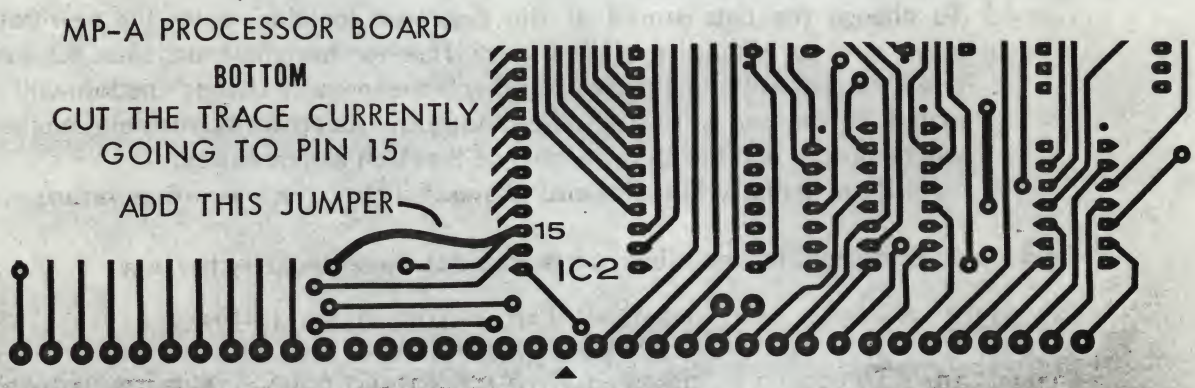
SWTBUG® is a permanently stored program and cannot be erased or lost by either a loss of power or user program error. It is always resident in the computer while power is ON and need never be loaded into the machine. Subroutines within the ROM are documented and available to the user to simplify programming and conserve on the use of user RAM memory. Character input and output, string output and return to monitor are just a few subroutines available to the user.

SWTBUG® is a 1K byte program and is addressed high in memory, far above the amount of RAM memory required for most user programs. Since SWTBUG® does require a small amount of RAM memory for operation, a 128 byte scratchpad RAM has been implemented on the processor board so that no external user RAM memory is required for monitor operation. There is even enough extra room in this RAM so that short programs such as memory diagnostics can be loaded into and run from the scratchpad RAM without requiring any external user RAM memory. Extra care however must be exercised to avoid overstoring any memory locations required for proper monitor operation. Complete details on this are given later in this writeup. The SWTBUG® ROM is located from memory addresses E000 thru E3FF. The scratchpad RAM is located from memory addresses A000 thru A07F. Both components are physically located on the processor board and are functional any time the system is powered up. Whenever computer control is transfered from SWTBUG® to the user program, there are only four ways to get back into the SWTBUG® command mode. The first is to put a jump to the CONTRL entry point address within SWTBUG® as the last step of your program. The second is to incorporate a command or action within your program which transfers program control to the CONTRL entry point address within SWTBUG® . The third is to depress the front panel RESET button. The fourth is turn the computer OFF and then back ON again. The fourth method is rather drastic and wipes out all RAM memory data, it is only mentioned to let you know that the computer always powers up with the SWTBUG® monitor in the command mode.

SWTBUG® INSTALLATION

SWTBUG® is a MOS device and MOS integrated circuits are susceptable to damage by static electricity. Although some degree of protection is provided internally within the integrated circuits, their cost demands the utmost in care. Before opening and/or installing SWTBUG® you should ground your body and all metallic tools coming into contact with the leads, thru a 1M ohm ¼ watt resistor. The ground must be an "earth" ground such as a water pipe, and not the circuit board ground. As for connection to your body, attach a clip lead to your watch or metal ID bracelet. Make absolutely sure that you have the resistor connected between you and the "earth" ground, otherwise you will be creating a dangerous shock hazard. Avoid touching the leads of the integrated circuits any more than necessary when installing it, even if you are grounded. Static electricity should be an important consideration in cold, dry environments. It is less of a problem when it is warm and humid.

When using SWTBUG® with an MP-A processor card, one board change is necessary since SWTBUG® is a full 1K ROM and MIKBUG® was a ½K device, If this ROM is replacing MIKBUG®, remove MIKBUG® from its socket. On the back side of the MP-A board you will notice that pin 15 of IC-2 (ROM) is grounded. The land coming from pin 15 should be broken and a wire added as shown below.



SWTBUG® should now be installed in the socket for IC-2. Be sure to orient the ROM correctly when re-installing. The semicircle notch or dot should match with the MP-A board's component layout drawing.

When installing SWTBUG® in the MP-A2 processor board no board modifications are necessary. Follow the instructions supplied with the MP-A2 instruction set.

SWTBUG® OPERATION

The SWTBUG® firmware enables the computer to communicate with a terminal to perform various programming and debugging functions. SWTBUG® will communicate with a terminal via either a MP-C control interface or MP-S ACIA serial interface on I/O port 1. An optional MP-C interface can be installed on I/O port 0 for punch and load functions. Although SWTBUG® is essentially compatible with MIKBUG®, be sure to read the COMPATIBILITY section before running any programs written for MIKBUG®. Below is a detailed description of each SWTBUG® command.

RESET

Upon receiving a RESET command, as during power up, SWTBUG® will initialize the system to receive commands from a terminal. When the RESET button is pushed, control will transfer to location EØDØ of SWTBUG®. The RESET button should be used for exiting loops or malfunctioning programs. After resetting, the computer should respond with a carriage return, line feed and a $ sign. At this point, SWTBUG® is waiting for commands. If breakpoints are being used, the RESET function will not disable breakpoints. The BREAKPOINT function should be referenced for additional information.

MEMORY EXAMINE AND CHANGE M (addr)

The Memory Examine and Change function can be used to enter machine code programs and to display and/or change the contents of memory. The Memory Examine and Change function should be used as follows:

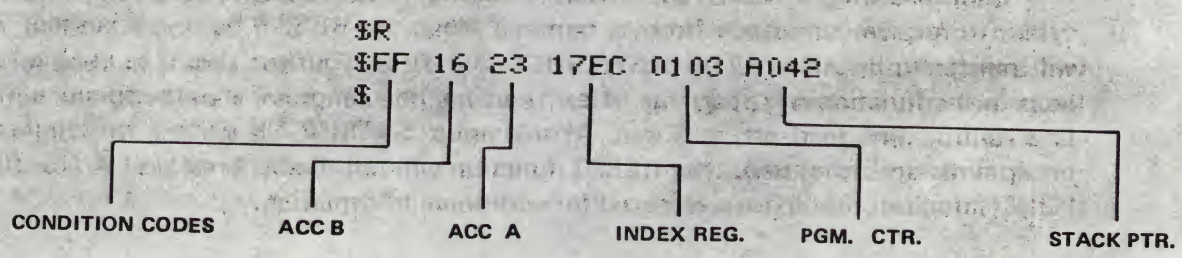
1.) Type M. The computer should echo the M and output a space.
2.) Type in the four digit hexadecimal address that you wish to examine and/or change. The computer should respond with a carriage return, line feed, $, the address and the data that is stored at this address.
3.) At this point the user has the option of advancing, either forward or backward, to the next memory location, or changing the data stored at the displayed address and advancing to the next location or of exiting the M function.
   a.) To display the next sequential address and data, a line feed or any character other than 0123456789ABCDEF :;↑=>? or a space or a carriage return may be entered. Any leading spaces that are entered will be ignored by the memory change function.
   b.) To display the next sequential address going backward from the present location, a ↑ should be entered.
   c.) To change the data stored at the displayed location, enter the new data, either with or without a leading space. If a non-hex value, such as a 3Q is entered the data will remain unchanged and the memory change function will be exited. If the data is unable to be changed (write protected memory, etc.) a ? will be output and the memory change function will be exited.
   d.) To exit the Memory Examine and Change function, type a carriage return.

Below is an example. The underlined parts are what was entered by the user.

```
$M 0100                 MEMORY LOCATION 0100 OPENED
$0100 00  .             DISPLAY NEXT LOCATION
$0101 BD  ___.          DISPLAY NEXT LOCATION - SPACES IGNORED
$0102 5D  .             DISPLAY NEXT LOCATION
$0103 C1  01            CHANGE CONTENTS TO 01
$0104 C9      23        CHANGE CONTENTS TO 23.  SPACES IGNORED
$0105 15  ^             READ PREVIOUS LOCATION
$0104 23  .             DISPLAY NEXT LOCATION
$0105 15  3Q            ENTER NON-HEX VALUE
$M 0105                 SWTBUG CONTROL RESUMED. OPEN NEW LOCATION
$0105 15                EXIT BY HITTING CARRIAGE RETURN
$M E000                 OPEN ANOTHER LOCATION
$E000 FE  567           ATTEMPTED TO CHANGE WRITE PROTECTED MEMORY
$                       SWTBUG CONTROL RESUMED
```

REGISTER DUMP FUNCTION R

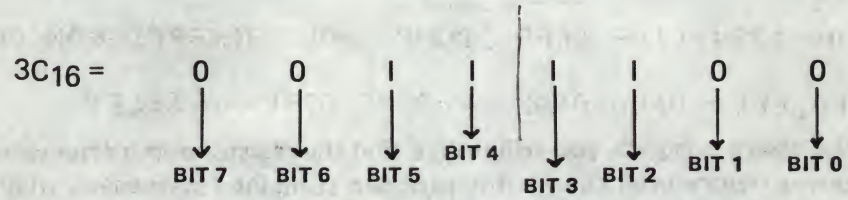
The R command will display the current contents of the MPU's pushdown stack. "Current" means the status of data stored on the stack immediately before SWTBUG® control is resumed. The following is a typical register dump:



The condition codes are as defined below:

BIT NO.	LABEL	CONDITION CODE
Ø	C	Carry—borrow
1	V	Overflow
2	Z	Zero
3	N	Negative
4	I	Interrupt mask
5	H	Half carry

In the above example the condition code of "3C" can be interpreted as follows:



Below are two examples of how the R command works. Assume that this small program was entered to change certain registers.

```
$M 0100
$0100 CE  8E  LOADS STACK POINTER TO 1000
$0101 12  10
$0102 34  00
$0103 86  CE  LOAD INDEX REGISTER WITH 1234
$0104 00  12
$0105 C6  34
$0106 FF  86  LOAD ACCUMULATOR A WITH 00
$0107 FD  00
$0108 08  C6  LOAD ACCUMULATOR B WITH FF
$0109 23  FF
$010A 67  7E  JUMP BACK TO SWTBUG CONTROL
$010B F7  E0
$010C 60  E3
$010D DD
AT THIS POINT THE STATUS WILL NOT BE PUSHED ON THE STACK
```

```
$R
$FF DC FC 6EFD 0100 A042  REGISTER DUMP BEFORE RUNNING PROGRAM
$G
$R
$FF DC FC 6EFD 0100 A042  NOTE R DUMP THE SAME AFTER RUNNING
$J 0100
$R
$FF DC FC 6EFD 0100 A042  THE SAME AFTER A JUMP
$M 010A                   AT THIS POINT THE JUMP TO SWTBUG CONTROL
$010A 7E  3F              IS REPLACED WITH A SWI.  NOTE THE VALUE
$010B E0                  OF THE REGISTERS AFTER THE NEXT DUMP.
$G
$F9 FF 00 1234 010A 0FF9  R DUMP GIVEN BY SWI
$R
$F9 FF 00 1234 010A 0FF9  DUMP SHOWS THE PROGRAM CHANGES
$
$FF DC FC 6EFD 0100 A042  R DUMP AFTER A RESET
```

In the above program you will notice that the register dump after running the program is the same as before even though the program contained statements that changed the processor's registers. The register dump did not reflect these changes because the new conditions were not pushed on the computer's stack. Note, however, the register dump did reflect the change when the last instruction was a software interrupt—a SWI instruction will push the processor's status on the stack and then display the contents of the registers.

CT-1024 CLEAR SCREEN COMMAND C

The **C** command outputs a home-up (10_{16}) and an erase to end of frame (16_{16}) control characters for the clearing of the screen on a SWTPC CT-1024 or equivalent terminal system.

GO TO USER'S PROGRAM FUNCTION G

Upon entering a **G** command, SWTBUG® will transfer control to the user's program by executing a RTI (return from interrupt) instruction. This effectively causes the computer to jump to the memory address stored in memory locations AØ48 and AØ49 in the SWTBUG® RAM. AØ48 contains the most significant byte and AØ49 the least significant byte of the memory address. If, for example, you wish to execute a program starting at Ø1ØØ, change AØ48 to a Ø1 using the **M** function and change AØ49 to ØØ. Typing a **G** will then cause the computer to execute the program whose starting address is Ø1ØØ. Upon entering a program using the **G** function, the stack pointer will be set at AØ49. The **G** function is also used to restart a program after a breakpoint has been moved. In this case AØ48 and AØ49 should not be changed. See the Breakpoint section for further information.

JUMP TO USER'S PROGRAM J(addr)

The **J** command will cause the computer to execute the program whose starting address is given in the entered address. The **J** command does not look at locations AØ48 and AØ49. The stack pointer will be set at AØ42 upon entering a program with a jump command. Example: J Ø1ØØ. If a non-hex character is entered, SWTBUG® control will resume.

ASCII TAPE PUNCH COMMAND P

The **P** command provides a means for storing the contents of specified memory on either cassette or paper tape. Normal output from the computer during a punch is thru either an MP-C or MP-S serial interface on I/O port 1, but a MP-C interface on I/O Ø can be selected. (See the description of the **O** command.) To use the **P** command, the upper and lower limits of the range to be punched must first be stored in locations AØØ2-AØØ5 of the

SWTBUG® RAM. If you wanted to punch from addresses 0123 to 4567 (inclusive) you would use the memory examine and change functions to set computer memory as follows.

A00Ø2 ⟶ Ø1 MOST SIGNIFICANT BYTE OF LOWER ADDRESS
A00Ø3 ⟶ 23 LEAST SIGNIFICANT BYTE OF LOWER ADDRESS
A00Ø4 ⟶ 45 MOST SIGNIFICANT BYTE OF UPPER ADDRESS
A00Ø5 ⟶ 67 LEAST SIGNIFICANT BYTE OF LOWER ADDRESS

Typing P would turn the punch on and output the specified memory data. A sample punch output is as follows:

```
$M A002
$A002 02  01      MSB OF LOW ADDRESS
$A003 72  00      LSB OF LOW ADDRESS
$A004 EF  01      MSB OF HIGH ADDRESS
$A005 00  20      LSB OF LOW ADDRESS
$A006 5F          TAPE PUNCH COMMAND
$P
S11301008E1000CE12348600C6FF3FE0E3DD005DB2
S1130110809E05160F73A8201F500FFC79771D1F2
S104012000DA
$
```

```
S11301008E1000CE12348600C6FF3FE0E3DD005DB2
```

START OF BLOCK CHARACTER — BYTE COUNT (HEX) — BLOCK STARTING ADDRESS — DATA — CHECKSUM

The S1 at the start of the block is used to tell the load routine that valid punch data follows. Each punch block must begin with the S1. The 13_{16} is the number of bytes that follow in the block. In this case two bytes are required for the starting address of the block (Ø1 and ØØ), $1Ø_{16}$ bytes are required for the data (8E1Ø ØØ CE 12 34 86 ØØ C6 FF 3F EØ E3 DD ØØ 5Ø) and one byte is required for the checksum (B2). The checksum is generated by adding the complement of the start of block address and the data, 8 bits at a time. At load-in time another checksum is generated by the load routine which must match with the one generated at punch time.

As the punch begins a PUNCH ON (12_{16}) control character will be output to the punch device. If a MP-C control interface is the selected interface, un-used lines on the PIA will be strobed to turn on the punch function of a SWTPC AC-30 or equivalent tape interface. (See the PIA Strobing section for more information.) When the punch is completed, a PUNCH OFF (14_{16}) control character will be output and another PIA line will be strobed. User control is then returned to SWTBUG®. To complete the tape making procedure you will have to enter the end of tape command described below.

END OF TAPE COMMAND E

The E command will punch the contents of the program counter (AØ48-AØ49) and an S9 to tape. The S9 is decoded by the load routine as an "end of tape" marker. For example, if you wish to save a program that resides from ØØØØ to ØØØF and whose starting address is ØØØ5, you would perform the following sequence-

```
$M A002
$A002 FC  00
$A003 3E  00
$A004 A0  00
$A005 49  0F
$A006 4C
```

```
$M A048
$A048 01   00
$A049 03   05
$A04A F4
$P
S11300000A0501001EF023FF01A01B351B37022443
$E
S105A04800050D59
$
$
```

AØ48 and AØ49 are automatically transferred to AØØ2-AØØ5 and punched to tape by the E command. A short delay follows the S9 to allow clean load-ins on cassette tape. Appropriate punch on/off commands are automatically sent.

TAPE LOADER FUNCTION L

The L function is used to load either a MIKBUG® or SWTBUG® formatted program from either paper or cassette tape. To use the L function, first set up the tape in the loading device and type L. As with the punch command, a L function will load from an MP-S or MP-C on I/O port 1 or if selected a MP-C on I/O port Ø. A READER ON (11_{16}) control character is output at the beginning of the load and terminal echo is disabled. Again the PIA is strobed for use with a tape interface. When the S9 end of tape marker is read, control will return to SWTBUG® . If a checksum error is detected, a ? will be printed and control will return to SWTBUG®. The load routine verifies that the data being loaded is actually stored in the correct memory locations. If, for some reason, a byte is not stored (bad memory, etc.) the computer will respond with a ?. To abort the load function the RESET switch can be pressed. Before attempting to load a SWTPC binary formatted tape, be sure to read the COMPATIBILITY section.

OPTIONAL PORT COMMAND O (not zero)

The O (not zero) command enables the user to load from or punch to a MP-C control interface plugged on to port Ø. To use the O command, type O followed by the desired option P, E or L (punch, end of tape or load). The same rules apply for using the P, E and L functions as described earlier. The O command will not support an ACIA type serial interface on port Ø.

NOTE: When using a MP-C interface on port Ø, a RESET will not, automatically turn off the reader and punch as is done when installed on port 1. This makes it impossible to create or load a binary formatted tape from an MP-C interface installed on I/O port Ø using existing binary load and punch programs.

SOFTWARE BREAKPOINTS B(addr)

The B command enables the user to enter breakpoints (software interrupts) in a program for debugging purposes. Breakpoints enable a program to be stopped at any point for register examination, memory changing, etc.

To use the breakpoint function, first load in your program and set up AØ48 and AØ49 to the starting address of the program. Type B and then enter the address where you want to set the first breakpoint. SWTBUG®will store the data that was at this address and replace it with a 3F (software interrupt). Typing G for Go to User Program will start program execution. When the program reaches the 3F, execution will stop and the processor's registers will be displayed. The B function can be used again to move the breakpoint to a new location—SWTBUG® automatically replaces the data at the old breakpoint location with the original instruction. A G should be used to re-start the program—**do not** reset AØ48 and AØ49—SWTBUG® automatically pulls the restart location from the stack. To remove breakpoints, type B followed by a carriage return. Breakpoints should **always** be removed in this way after using the breakpoint function. If, while using breakpoints, the system is reset, the correct location on the stack will be lost. After resetting, the program counter addresses (AØ48, AØ49) should be re-initialized to the beginning of the program and execution restarted. A

previously set breakpoint will remain and may be changed or removed as described earlier. If, when using the B command, a non-hex value is entered the previous breakpoint will be removed and SWTBUG® control will resume.

There are several things that one must be aware of when using breakpoints to insure proper operation.

1.) The breakpoint function uses the same locations as do vectored software interrupts; therefore, vectored software interrupts should not be used with breakpoints.

2.) The SWI jump location, AØ12, will be set to E124 when breakpoints are not in use, as after power up, and will be set to E123 when breakpoints are in use. This location serves as a pointer to tell the computer what to do when a 3F is seen. The RESET button will **not** re-set this location to the non-breakpoint state. The breakpoint-activated state can **only** be exited by typing **B** followed by a carriage return. If you are using breakpoints in a program that "bombs out" and you hit the RESET switch, you must clear the present breakpoint before going on to another program. If this is not done before a new program is loaded in, the first time the **B** command is used one byte of the new program will be replaced by the stored byte from the last program.

3.) Do not set a breakpoint to an address where a breakpoint is already set. Doing so will cause the computer to lose the original program data.

DO NOT	THIS IS OK
B 1377	B 1377
G	B ØlØØ
B 1377	B1377
	B carriage return

4.) The breakpoint routine uses SWTBUG RAM locations AØ14—AØ16; therefore, programs which use these three bytes should not be used in conjunction with breakpoints.

DISK BOOT D

SWTBUG® contains the boot necessary to initialize a SWTPC MF-68 disk system. Typing D will transfer control to the disk operating system, (if attached). If D (is accidentally typed with no disk attached, the reset button must be pressed. Since the disk boot contains no error detection, it may need to be typed more than once to do a boot.

JUMP TO PROM PROGRAM Z

Typing Z will transfer control to a program stored in PROM (if applicable) whose starting address is at CØØØ. Typing **Z** is the equivalent of typing J CØØØ.

BYTE SEARCH F (high address) (low address) (byte)

The **F** (find) command will search memory from the specified low address to the high address, inclusive, and will display all memory locations containing the byte specified. For example, to find all memory locations between ØlØØ and Ø2ØØ that contain 8E, the following command should be used: F Ø2ØØØlØØ8E. Note that no spaces may be used between addresses and that the high address goes first.

```
$F 020001008E
```

If a non hex value is entered, SWTBUG control will resume.

VECTORED SOFTWARE INTERRUPTS

Normally when encountering a SWI (3F) instruction, the computer will display the processor's registers and SWTBUG® control will be resumed. If desired, the 3F command can be vectored to anywhere in memory, just like the NMI and IRQ interrupts. To use the vectoring capability simply store the service routine address at location AØ12 - AØ13 in the SWTBUG® RAM. When a 3F is encountered, processor control will be transferred to the memory address stored in AØ12-AØ13. Note: each time the system is RESET, AØ12 will be reset to the location of the register dump routine. This means that any program which uses vectored SWI's should set up this location each time it is executed. If the location you wish to vector to is 1ØDØ, for example, the following statements at the beginning of the program will set up the vector correctly:

```
LDX # $1ØDØ    LOAD VECTOR ADDRESS
STX  $AØ12     STORE VECTOR
```

Vectored software interrupts should not be used in conjunction with breakpoints since the breakpoint routine uses locations AØ12 - AØ13.

VECTORED INPUT/OUTPUT

If desired, input and output can be vectored to a MP-S or MP-C interface on ports other than # 1. Locations AØØA - AØØB contains the port address that the subroutines INEEE and OUTEEE use for inputting and outputting characters. To use vectored input/output your program must store the desired I/O address in AØØA -AØØB before any I/O is done. Below is a list of I/O address assignments for each port:

PORT	ADDRESS
Ø	8ØØØ
1	8ØØ4
2	8ØØ8
3	8ØØC
4	8Ø1Ø
5	8Ø14
6	8Ø18
7	8Ø1C

The program statements that would set up the correct port would be as follows:

```
LDX  # $8Ø18     I/O on port 6
STX    $AØØA     Store
```

SWTBUG® will look at the port and will self-configure for either a MP-C or MP-S type interface.

NOTE: Any time that SWTBUG®'s control sequence is initiated or when the RESET button is pushed, the I/O address will be reset to port # 1. Therefore complete SWTBUG® monitor control cannot be moved to another port.

USING NON-MASKABLE INTERRUPTS

Using non-maskable interrupts is very similar to using vectored software interrupts. A non-maskable interrupt will occur whenever the NMI line on the computer's bus is grounded either through hardware or by an ACIA or PIA. When the NMI is seen, processor control will be transferred to the location stored in AØØ6 and AØØ7. For example if an NMI service routine is desired at location 1ØØØ the following statements should be used at the beginning of your program to set up the correct NMI jump address.

```
LDX # $1ØØØ
STX   $AØØ6
```

USING MASKABLE (IRQ) INTERRUPTS

Using regular maskable interrupts is the same as using non-maskable interrupts except that when the IRQ line is grounded processor control will jump to the address stored in AØØØ and AØØ1. The computer will only respond to the interrupt if the processor's interrupt mask bit I is Ø. A CLI instruction at the beginning of your program will insure this condition.

PIA STROBING

Use of the Control Interface for Read/Punch–On/Off Decoding

SWTBUG® software contains subroutines to send out pulses to unused pins of the PIA integrated circuit on the MP-C serial control interface that can be used for automatic reader/punch controls. These pulses can be used if you are using a SWTPC AC-30 cassette interface and a terminal in which access to the control command decoding is denied.

If you intend to use the read/punch control logic output on the MP-C control interface board, make the following connections from the indicated pins of IC1 on the MP-C control interface board to the specified pins of a twelve pin male connector shell. The connector pinning shown below is correct for a SWTPC AC-30 cassette interface and will need modification for other units. Be sure to make the wires long enough to reach your AC-30 where the connector will be plugged. If you have access to your terminal's 16X baud rate clock, the terminal's clock bus should be broken and the 16X clock IN and OUT lines brought out to the same connector.

MP-C IC1 pin 7 (read on)	12 pin male shell female pin 1
MP-C IC1 pin 4 (punch on)	12 pin male shell female pin 2
MP-C IC1 pin 6 (read off)	12 pin male shell female pin 3
MP-C IC1 pin 5 (punch off)	12 pin male shell female pin 4
Terminal's 16X clock OUT	12 pin male shell female pin 5
Terminal's 16X clock IN	12 pin male shell female pin 6
MP-C ground	12 pin male shell female pin 12

These signals are low going pulses and are about 15 microseconds wide. They are not buffered and should drive a maximum of only one standard TTL load.

PIA strobing will work only on SWTBUG®'s L, P and E functions. Strobing is not supported in BASIC and some other SWTPC software.

OPERATING THE MP-A2 PROCESSOR BOARD AT BAUD RATES HIGHER THAN 1200 BAUD

The MP-S Serial Interfaces available for the SWTPC 6800 Computer System are capable of operating at baud rates up to 9600 baud. Although baud rate clocks for 110, 150, 300, 600 and 1200 baud are generated, buffered and fed onto the mother board by IC4 of the MP-A2 board, clocks for additional baud rates are also available from IC4 as well. The table below gives the baud rate and respective output pin number of IC4.

BAUD RATE	MP-A2 IC4 pin
75	9
200	6
1800	15
2400	3
3600	16
4800	2
7200	17
9600	1

To use the selected clock, run an insulated jumper between the specified pin and pin 13 of IC10 on the MP-A2 board. Run another insulated jumper between pin 12 of IC10 and either the UD1 or UD2 bus connections points at the connector edge of the MP-A2 circuit board. IC10 is a low power TTL buffer which must be inserted between the baud rate clock generator and the mother board bus. Since user defined lines UD1 and UD2 are carried on just the 50-pin main board bus and lines UD3 and UD4 are carried on just the 30-pin interface board, it will be necessary to jumper two of the buses together to provide the selected baud rate clock on the interface card bus. Each serial interface card to be operated with the selected buad rate clock will have to be jumpered so its clock is acquired from the selected user defined line rather than one of the five original baud rate clocks already present.

OPERATING THE MP-A PROCESSOR BOARD AT BAUD RATES HIGHER THAN 1200 BAUD

When using the MP-S serial interface with an MP-A processor board, baud rate clocks for up to 9600 baud are available from the baud rate generator on the MP-A processor board. The table below shows the baud rates available and from which pin of IC4 on the MP-A board they are derived. These 16X baud rate clocks are best fed back to the interface boards via the user defined lines provided on the mother board. These baud rates of course are in addition to the 110, 150, 300, 600 and 1200 baud rate clocks already provided on the mother board.

BAUD RATE	MP-A ICA Pin
75	9
200	6
1800	15
2400	3
3600	16
4800	2
7200	17
9600	1

COMPATIBILITY

Although SWTBUG® has been written to be as compatible as possible with MIKBUG® and with software supplied by SWTPC, it can never be completely MIKBUG® compatible. All major subroutines of SWTBUG® are address and function compatible with MIKBUG®, but if you have a program that enters into the middle of a MIKBUG® routine for some reason, program modifications will be necessary. The following is a list of the MIKBUG® compatible subroutines and strings along with their entry point addresses.

EØ4Ø	LOAD19	EØC8	OUT 4HS
EØ47	BADDR	EØCA	OUT2HS
EØ55	BYTE	EØDØ	START
EØ75	OUTCH	EØE3	CONTRL
EØ78	INCH	E19C	MCLOFF
EØ7B	PDATA2	E19D	MCL
EØ7E	PDATA1	E1AC	INEEE
EØBF	OUT 2H	E1D1	OUTEEE

If any doubt exists as to the compatibility of a particular program, it should be disassembled and any references to memory locations EØØØ -E1FF be verified.

Since SWTBUG® is more complex than MIKBUG®, more RAM area must be used in the 6810 SWTBUG® RAM. If vectored software interrupts and breakpoints are not being used, the area from AØ14 to AØ33 and from AØ4A to AØ7F can be used for small, temporary programs such as memory diagnostics. Note that some programs written for MIKBUG® use locations AØ34 - AØ36—these locations are not available for use in SWTBUG®.

LOADING BINARY TAPES THRU SWTBUG®

SWTBUG® was written to accept the binary formatted tapes supplied by SWTPC. These tapes include 4K BASIC, 8K BASIC, CORES and DESEMBLER. When loading these tapes the following rules must be followed:

1.) The tape reading device (AC-30, etc.) must be locked in the read on mode during the binary load.

2.) Binary tapes must be loaded in thru port #1, the control port. The optional load from port Ø command is **not** supported in binary. You may load in ASCII however.

3.) When using a PIA type interface to load binary tapes, the unused lines used for reader/punch on/off strobing are not activated.

NOTE: This does not mean that SWTBUG® is equipped with a binary loader—only certain SWTPC binary tapes that contain a special binary loader (in ASCII) will work correctly.

To load the tape simply follow the instructions given for loading an ASCII tape, but keep the reader locked on.

SPECIAL NOTES ON USING AN ACIA AND PROGRAM MODIFICATIONS

Many available 6800 programs written for MIKBUG® assume that a PIA' type MP-C control interface is being used and may address this port directly. When using an ACIA type interface, these references need to be changed. For example, some programs, such as BASIC and CO-RES, poll the PIA periodically to see if a character has been typed in. This is done in order to kick out of a loop or a print sequence. (BASIC uses CTL.C and CO-RES uses CTL.X.) The source statements that do this usually take the following form:

```
B6 8ØØ4          LDA A PIAD        LOAD A FROM DATA REG.
2B Ø3            BMI PRINT         BRANCH IF NOT CHAR. SEEN
7E XX XX         JMP READY or RESTART
           PRINT REMAINDER OF SEQUENCE
```

To change to a MP-S serial interface, this code can sometimes be replaced as follows;

```
LDA A PIAD ——→  ASR A             SEE IF CHAR. LOADED
BMI   PRINT ——→ BCS PRINT         BRANCH IF CHAR. INPUT
                JMP READY
           PRINT REMAINDER OF SEQUENCE
```

Before modifying any programs on your own, you should have a working knowledge of SWTBUG®'s ACIA input routine, ACIA operation, and your particular program. The following is a list of patches to some SWTPC supplied programs.

BLKJAK — SWTPC 6800 Black Jack Program

LOCATION	DATA
Ø27Ø	7E Ø64A
Ø647	7E Ø26D
Ø64A	B6 EØØ8
Ø64D	27 Ø8
Ø64F	B6 8ØØ4
Ø652	2B F3
Ø654	7E Ø275
Ø657	B6 8ØØ4
Ø65A	47
Ø65B	24EA
Ø65D	2Ø F5

With the above modifications BLKJAK will be compatible with either an ACIA or PIA type interface.

CO-RES Ver. 1.0 and 1.01 ACIA Modifications

LOCATION	DATA
1682	47
1683	24 Ø2
1685	2Ø AØ

BASIC 8K and 4K up to an including Ver. 2.Ø cannot be modified for ACIA operation. Later versions should be purchased.

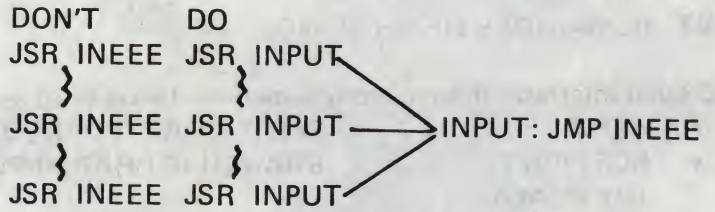
GENERAL RULES FOR PROGRAM WRITING

Although for a user program to be functional it need only work with the exact system it was written for, following a few simple rules reduces program modifications for 6800 systems using other monitors. Following these rules will make your programs more professional and versatile. Some general guidelines are as follows:

1.) Minimize the number of references made to the ROM.

2.) Do not use strange, in-between SWTBUG® addresses. Generally only the routines BADDR, BYTE, PDATA1, INHEX, OUT4HS, OUT2HS, CONTRL, INEEE and OUTEEE should be used.

3.) For large programs, vector I/O through a jump instruction for ease of change to match other I/O packages. Example:



4.) Try not to use the SWTBUG® RAM any more than necessary. With the exception of using it as stack storage and memory diagnostics, there is no real reason to use the SWTBUG® RAM area.

5.) Define the stack area at the beginning of the program. Example: Start LDS #$ AØ42. Relocating the stack location to AØ42 at the beginning of each of your programs will prevent you from having to reload the program counter addresses AØ48 and AØ49 each time you RESET and restart your program.

6.) Most programs should have a provision for exiting them without hitting the RESET button. A jump to CONTRL (7E EØE3) instruction in your program will cause SWTBUG® control to resume when executed.

MEMORY DIAGNOSTICS

The earlier memory diagnostics ROBIT, MEMCON and CDAT supplied by SWTPC were compatible only with MIKBUG®. The new versions ROBIT 2, MEMCON 3 and CDAT 2 are compatible with both MIKBUG® and SWTBUG®.

PROGRAM DESCRIPTION

Although the source listing of SWTBUG® is well commented, the following subroutine by subroutine description should be of use to those who wish to gain the maximum advantage of its routines.

TEMPORARY STORAGE LOCATIONS

IRQ (A000) This location is used by the standard IRQ interrupt request feature. When an interrupt is generated, processor control will jump to the location stored in IRQ.

BEGA (A002) This location is where the beginning address is stored for the punch and end of tape routines.

ENDA (A004) This location is where the ending address is stored for the punch and end of tape routines. It is also used by the byte search routine.

NMI (A006) NMI is used by the non-maskable interrupt (NMI) function. When an NMI is generated, processor control will jump to the location stored in NMI.

SP (AØØ8) Temporary storage location for the stack pointer. SP is used in the register dump subroutines and by the breakpoint function.

PORADD (AØØA) This location contains the port address used for SWTBUG®'s I/O routines.

PORECH (AØØC) This byte tells SWTBUG®'s input routines whether or not to echo.

XHI (AØØD) Temporary index register storage used by numerous routines.

XLOW (AØØE) Temporary index register storage used by numerous routines.

XTEMP (AØ1Ø) Temporary index register storage for input and output routines.

SWIJMP (AØ12) When a SWI instruction is encountered, processor control will transfer to the location stored in SWIJMP.

BKPT (AØ14) Temporary breakpoint address storage.

BKLST (AØ16) Temporary data storage for the breakpoint routine.

TW (AØ44) Temporary storage location for load/punch.

TEMP (AØ46) Temporary storage location for punch and load.

BYTECT (AØ47) Temporary storage location for load/punch.

SWTBUG® SUBROUTINE AND TEXT STRING DESCRIPTION

IRQV (EØØØ) This is the entry point for regular IRQ interrupts. Processor control is given to the service routine whose address is stored in IRQ.

JUMP (EØØ5) This is the service routine for the J command. BADDR is used to input the address and a jump then occurs to the correct address.

CURSOR (EØØ9) Home-up and erase to end of frame characters for CT 1024.

LOAD (EØØC) Load is the ASCII loading routine. Load uses a number of other SWTBUG® subroutines.

BADDR (EØ47) BADDR is a subroutine to input a 4-digit hexadecimal number, such as 137D, from the control terminal. BADDR uses the subroutines BYTE, INCH and INHEX and uses temporary storage locations XHI, XLOW, CKSM, both accumulators and the index register. When BADDR is called it will look for four hex numbers to be entered from the terminal. If a non-hex value, such as H, is entered, SWTBUG control will resume. If all characters entered are valid hex, the results will be stored in XHI, XLOW and the index register. Accumulator A will contain of XLOW. If 137D is entered the results will be as follows-

ACC A	7D
ACC B	CKSM
IXR	137 D
XHI	13
XLOW	7D

CKSM and ACC B are used internally to generate a check sum for the PUNCH routine.

BYTE (EØ55) BYTE is similar to BADDR, but inputs only two hex characters from the terminal to generate one 8-bit byte equivalent. BYTE uses the subroutines INHEX and INCH, temporary storage locations CKSM and both accumulators. If a non-hex value is entered, SWTBUG® control will resume. When BYTE is called as a subroutine, the computer will wait for two hex characters to be entered thru the control port. If a 3C is entered, the results will be as follows:

ACC A	3C
ACC B	CKSM
IXR	UNCHANGED
CKSM	Prior CKSM + check sum generated inside BYTE

OUTHL (EØ67)
OUTHR (EØ6B) These subroutines are used by OUT2HS and OUT4HS to output hexadecimal numbers.

OUTEEE,
OUTCH
OUTEE1
(E1D1) This is the character output routine used by PDATA1, OUT4HS, OUT2HS and most programs written for SWTBUG®/MIKBUG® to output one character from the computer to the control port (I/O # 1). OUTEEE, OUTCH and OUTEE1 are all functional equivalents—OUTEE1 is the main output routine with OUTCH and OUTEEE being jumps to OUTEE1. When using this routine, OUTEEE (E1D1) sould be used to maintain compatibility with MIKBUG® systems.

To use OUTEEE the character to be output should be placed in the A accumulator in its ASCII form. To output the letter A on the control terminal, the following program could be used.

```
LDA     A# $41
JSR     OUTEEE
```

The processor's registers are affected as follows.

ACC A	changed internally
ACC B	not affected
IXR	not affected

OUTEEE is an 8-bit output routine and does not generate a parity bit.

INEEE,
INCH,
INEEE 1
(E1AC) The locations are all functionally equivalent to SWTBUG®'s character input routine. This routine will look for one character from the control terminal (I/O # 1) and store it in the A accumulator. Once called, INEEE will loop within itself until a character has been input. Anytime input is desire, the call JSR INEEE should be used.

INEEE automatically sets the 8th bit to Ø and does not check for parity. When using INEEE the processor's registers are affected as follows:

ACC A	loaded with the character input from the terminal
ACC B	not affected
IXR	not affected

INCH8 (E1F6) INCH8 is functionally the same as INEEE except that the 8th bit is not set to Ø. This subroutine should be used whenever full 8-bit input is desired, such as in binary loader programs.

INHEX (EØAA) INHEX is the subroutine used by BYTE and BADDR that will input one hexadecimal character from the control terminal. If a non-hex character is entered, SWTBUG® control will resume. If a hex character, such as an E is entered, the results will be as follows:

ACC A	ØE
ACC B	not affected
IXR	not affected

PDATA1 (EØ7E) PDATA1 is the subroutine used to output a string of text on the control terminal. PDATA1 will start outputting data that is pointed to by the index register and will continue until a Ø4 is seen. For example, if you wanted to print HELLO on the terminal the following could be used.

```
        ORG $1ØØ
START   LDX # TEXT
        JSR  PDATA1
        JMP  CONTRL
TEXT    FCB  $ØD, $ØA
        FCC  /HELLO/
        FCB  4
        END
```

The accumulator and register status after using PDATA1 is as follows:

ACC A	Changed during the operation
ACC B	UNCHANGED
IXR	Contains the memory location of the Ø4

CHANGE (EØ88) CHANGE is SWTBUG®'s memory examine and change function. Change uses a number of other SWTBUG® subroutines.

OUT4HS (EØC8) OUT4HS is used to output a four-digit (16-bit) hexadecimal number onto the control terminal. The address of the most significant byte to be output should be loaded into the index register before calling OUT4HS. For example, to output the 16-bit hex number stored in memory locations 1ØØØ and 1ØØ1 whose most significant byte is in 1ØØØ while the least significant byte is in 1ØØ1 the following sequence should be used:

```
LDX  # $1ØØØ
JSR  OUT4HS
```

If location 1ØØØ contained a hex 3C and 1ØØ1 contained a hex ØB, an ASCII, 3CØB would be displayed on the screen when OUT4HS is called. Remember memory data is handled in hex but must be output as ASCII characters which have a different hex value than those stored in the computer's memory. The registers are affected as follows:

ACC A	changed during the operation
ACC B	unchanged
IXR	Incremented by two. (1002 in this example)

OUT2HS (EØCA) OUT2HS is similar to OUT4HS, but outputs only two hex characters (one byte). For example, to display the byte stored at 2ØØØ the following sequence would be used:

```
LDX    #$2ØØØ
JSR    OUT2HS
```

An ASCII 6C would be output to the control terminal if location 2ØØØ contained a hex 6C. The registers are affected as follows:

ACC A	changed in the operation
ACC B	unchanged
IXR	Incremented by one (2ØØ1 in this example)

OUTS (EØCC) OUTS is a subroutine that outputs one space to the control terminal.

START (EØDØ) START is the beginning of a sequence of steps that initialize the system during power up or reset. First, the stack pointer is set to be AØ42 and is stored in SP. Next, an FF is stored in location AØ43. This sets the interrupt mask bit in the stack so that when a G command is given the processor will not respond to

interrupts until told to do so. The type of port being used (ACIA or PIA) is then determined by initializing it as a PIA and looking to see if this worked. If not, an ACIA is assumed and the proper ACIA initialization routine, AL2, is selected. The program then goes to CONTRL sequence.

CONTRL (EØE3) This MIKBUG® equivalent sequence again resets the stack to AØ42. PORECH is cleared to enable echo and the subroutine SAVGET is selected to get the correct port number and type. Next, the routines PNCHOF and RDOFF generate punch and reader off commands. A carriage return, line feed, erase to end of line (15_{16}) and a $ is then output to the control terminal. At this point SWTBUG® is ready for command input.

SFE1 (E124) SFE1 is the entry point for non user-vectored software interrupt instructions. If vectored software interrupts are selected, a jump is executed to the proper location. If breakpoints are in use the stack pointer is not changed, the processor's registers are displayed and SWTBUG® is instructed to look for the next command. If neither breakpoints or vectored software interrupts are selected a register dump occurs and the CONTRL sequence is initiated.

PRINT (E13Ø) PRINT is the routine that actually does the dumping of the processor's registers.

LOOK (E173) LOOK is the routine that inputs a character from the terminal and jumps to the appropriate location in SWTBUG® if it is a valid command.

SFE (E18B) Entry point for software interrupt instructions.

S9 (E19Ø) S and 9 string for the end of tape routine.

MTAPE1 (E193) This is the character string containing a carriage return, line feed, erase to end of line, four nulls, a S1 for tape control and Ø4 for PDATA1 control.

MCL (E19D) This string contains a carriage return, line feed, three nulls, a $ and a Ø4 for PDATA1 control.

E1A5 (E1A5) E1A5 is a special entry location which is used by the binary load routine on some SWTPC binary tapes.

NMIV (E1A7) This routine fetches the correct jump location for a NMI.

SEARCH (E1AE) Byte searching routine.

GOTO (E1DØ) GOTO contains the RTI instruction that is used by the G command to execute a user program.

SAVGET (E1D3) This routine saves the index register in XTEMP and tests for the appropriate interface location and type. The index register is then loaded with the address of this interface.

ISACIA (E1D9) ISACIA is the routine that sees if an ACIA or PIA is present.

BILD (E1F3) BILD is a special sequence of increment stack pointer instructions used only by the binary loader on some SWTPC binary tapes.

ACIAIN (E1FF) This is the ACIA input routine. PORECH is polled for the desired echo/don't condition. The character in the A accumulator is then output, the index regis-

ter and B accumulator restored and an RTS instruction executed by the RES routine.

ACIOUT (E212) This is the ACIA output routine which outputs the character in the A accumulator, and recovers the B accumulator and index register.

IN1 (E223) IN1 is the PIA input routine which inputs a character from the control terminal and stores it in the A accumulator. The correct echo/non echo condition is selected and the B accumulator and index register are restored.

IOUT (E24Ø) IOUT is the PIA output routine which outputs the character in the A accumulator.

OPTL (E269) OPTL is the service routine that sets up PORECH for I/O on port Ø. The appropriate command P, E or L is then selected.

PIAECH (E27D) This routine disables the echo on a PIA type interface.

PIAINI (E284) This routine is used to initialize PIA type interfaces.

DELAY (E2Ø2) DELAY is a general purpose delay loop. If desired, the index register can be pre-loaded with a number other than FFFF for shorter delays. The entry point in this case is DELAY1 (E2C5).

CLEAR (E2CC) This routine generates a home up, erase to end of frame command for SWTPC CT 1024 and similar terminal systems.

BREAK (E2D9) BREAK is the routine that is used to enter software breakpoints. First, the address is input by BADDR. If breakpoints were previously in use the data is replaced at the previous breakpoint address and the new breakpoint is inserted.

PNCHS9 (E31E) This routine selects AØ48 and AØ49 as the beginning and ending address for the punch routine and punches their contents. A S9 is then punched to tape followed by a short delay.

RDON (E334)
RDOFF (E347)
PNCHON (E34D)
PNCHOF (E353) These subroutines are used to turn a reader/punch on and off. At the beginning of each routine the A accumulator is loaded with the ASCII value of the particular function that is normally decoded by a Teletype® or other terminal system. The B accumulator is then loaded with a special bit pattern that will cause a predetermined line to be toggled on PIA type interfaces. ACC A is then output using OUTEEE and if a PIA type interface is being used the proper PIA line is strobed by the STROBE routine. The proper line is determined by the contents of ACC B. The subroutine RDON also clears the location PORECH and re-configures PIA type interfaces to be sure that the echo function of the character input routine is disabled. Both accumulators and the index register are used and are not retained.

ACC A	ACC B	FUNCTION
11	2Ø	Reader ON
13	1Ø	Reader OFF
12	Ø4	Punch ON
14	Ø8	Punch OFF

STROBE (E357) This is the routine that actually generates the pulses on the un-used lines of a PIA type interface for reader/punch control.

PUNCH (E376) PUNCH is the ASCII punching routine of SWTBUG®. PUNCH consists of several parts and uses various SWTBUG® subroutines.

TABLE (E3D1) This is the command table used by the lookup routine. The table is arranged in three byte blocks. The first byte is the ASCII value of the command. The next two bytes are the address of the routine that will service the command.

```
PAGE  001   SWTBUG


00010                       NAM     SWTBUG
00020                *      VERSION 1.00

00040                       OPT     O
00050                ********************************************
00060                *REPLACEMENT FOR MIKBUG ROM
00070                *FOR SWTPC 6800 COMPUTER SYSTEM
00080                *COPYRIGHT 1977
00090                *SOUTHWEST TECHNICAL PROD. CORP.
00100                *AUGUST, 1977
00110                ********************************************


00140 A000                  ORG     $A000
00150 A000 0002     IRQ     RMB     2       IRQ POINTER
00160 A002 0002     BEGA    RMB     2       BEGINNING ADDR PNCH
00170 A004 0002     ENDA    RMB     2       ENDING ADDR PNCH
00180 A006 0002     NMI     RMB     2       NMI INTERRUPT VECTOR
00190 A008 0001     SP      RMB     1       S HIGH
00200 A009 0001             RMB     1       S LOW
00210 A00A 0002     PORADD  RMB     2       PORT ADDRESS
00220 A00C 0001     PORECH  RMB     1       ECHO ON/OFF FLAG
00230 A00D 0001     XHI     RMB     1       XREG HIGH
00240 A00E 0001     XLOW    RMB     1       XREG LOW
00250 A00F 0001     CKSM    RMB     1       CHECKSUM
00260 A010 0002     XTEMP   RMB     2       X-REG TEMP STGE
00270 A012 0002     SWIJMP  RMB     2       SWI JUMP VECTOR
00280      A044     TW      EQU     $A044   TEMPORARY STORAGE
00290      A046     TEMP    EQU     $A046   TEMPORARY STORAGE
00300      A047     BYTECT  EQU     $A047   BYTECT AND MCONT TEMP.
00310      8004     CTLPOR  EQU     $8004   CONTROL PORT ADDRESS
00320      C000     PROM    EQU     $C000   JUMP TO PROM ADDRESS
00330 A014 0002     BKPT    RMB     2       BREAKPOINT ADDRESS
00340 A016 0001     BKLST   RMB     1       BREAKPOINT DATA

00360 A042                  ORG     $A042
00370 A042 0001     STACK   RMB     1       SWTBUG STACK

00390 E000                  ORG     $E000

00410                *I/O INTERRUPT SEQUENCE
00420 E000 FE A000  IRQV    LDX     IRQ
00430 E003 6E 00            JMP     0,X

00450                *JUMP TO USER PROGRAM
00460 E005 8D 40    JUMP    BSR     BADDR
00470 E007 6E 00            JMP     0,X

00490 E009 10       CURSOR FCB      $10,$16,4  CT-1024 CURSOR CONTROL
      E00A 16
      E00B 04

00510                *ASCII LOADING ROUTINE
```

```
PAGE  002   SWTBUG


00520 E00C BD E334  LOAD    JSR     RDON      READER ON, DIS ECHO, GET P#
00530 E00F 8D 67    LOAD3   BSR     INCH
00540 E011 81 53            CMP A   #'S
00550 E013 26 FA            BNE     LOAD3     1ST CHAR NOT S
00560 E015 8D 61            BSR     INCH      READ CHAR
00570 E017 81 39            CMP A   #'9
00580 E019 27 29            BEQ     LOAD21
00590 E01B 81 31            CMP A   #'1
00600 E01D 26 F0            BNE     LOAD3     2ND CHAR NOT 1
00610 E01F 7F A00F          CLR     CKSM      ZERO CHECKSUM
00620 E022 8D 31            BSR     BYTE      READ BYTE
00630 E024 80 02            SUB A   #2
00640 E026 B7 A047          STA A   BYTECT    BYTE COUNT
00650                *BUILD ADDRESS
00660 E029 8D 1C            BSR     BADDR
00670                *STORE DATA
00680 E02B 8D 28    LOAD11  BSR     BYTE
00690 E02D 7A A047          DEC     BYTECT
00700 E030 27 09            BEQ     LOAD15    ZERO BYTE COUNT
00710 E032 A7 00            STA A   0,X       STORE DATA
00720 E034 A1 00            CMP A   0,X       DATA STORED?
00730 E036 26 08            BNE     LOAD19
00740 E038 08               INX
00750 E039 20 F0            BRA     LOAD11
00760 E03B 7C A00F  LOAD15  INC     CKSM
00770 E03E 27 CF            BEQ     LOAD3
00780 E040 86 3F    LOAD19  LDA A   #'?
00790 E042 8D 31            BSR     OUTCH
00800 E044 7E E2D4  LOAD21  JMP     RDOFF1

00820                *BUILD ADDRESS
00830 E047 8D 0C    BADDR   BSR     BYTE      READ 2 FRAMES
00840 E049 B7 A00D          STA A   XHI
00850 E04C 8D 07            BSR     BYTE
00860 E04E B7 A00E          STA A   XLOW
00870 E051 FE A00D          LDX     XHI       LOAD IXR WITH NUMBER
00880 E054 39               RTS

00900                *INPUT BYTE (TWO FRAMES)
00910 E055 8D 53    BYTE    BSR     INHEX     GET HEX CHAR
00920 E057 48       BYTE1   ASL A
00930 E058 48               ASL A
00940 E059 48               ASL A
00950 E05A 48               ASL A
00960 E05B 16               TAB
00970 E05C 8D 4C            BSR     INHEX
00980 E05E 1B               ABA
00990 E05F 16               TAB
01000 E060 FB A00F          ADD B   CKSM
01010 E063 F7 A00F          STA B   CKSM
01020 E066 39               RTS

01040 E067 44       OUTHL   LSR A             OUT HEX LEFT BCD DIGIT
01050 E068 44               LSR A
```

```
01060 E069 44            LSR A
01070 E06A 44            LSR A
01080 E06B 84 0F  OUTHR  AND A  #$F      OUT HEX RIGHT BCD DIGIT
01090 E06D 8B 30         ADD A  #$30
01100 E06F 81 39         CMP A  #$39
01110 E071 23 02         BLS    OUTCH
01120 E073 8B 07         ADD A  #$7

01140                    *OUTPUT ONE CHAR
01150 E075 7E E1D1 OUTCH JMP    OUTEEE
01160 E078 7E E1AC INCH  JMP    INEEE

01180                    *PRINT DATA POINTED TO BY X REG
01190 E07B 8D F8   PDATA2 BSR   OUTCH
01200 E07D 08            INX
01210 E07E A6 00   PDATA1 LDA A 0,X
01220 E080 81 04         CMP A  #4
01230 E082 26 F7         BNE    PDATA2
01240 E084 39            RTS             STOP ON HEX 04

01260 E085 7E E14A C1    JMP    SWTCTL

01280                    *MEMORY EXAMINE AND CHANGE
01290 E088 8D BD   CHANGE BSR   BADDR
01300 E08A CE E19D CHA51 LDX    #MCL
01310 E08D 8D EF         BSR    PDATA1   C/R L/F
01320 E08F CE A00D       LDX    #XHI
01330 E092 8D 34         BSR    OUT4HS   PRINT ADDRESS
01340 E094 FE A00D       LDX    XHI
01350 E097 8D 31         BSR    OUT2HS   PRINT OLD DATA
01360 E099 8D 31         BSR    OUTS     OUTPUT SPACE
01370 E09B 8D DB   ANOTH BSR    INCH     INPUT CHAR
01380 E09D 81 20         CMP A  #$20
01390 E09F 27 FA         BEQ    ANOTH
01400 E0A1 81 0D         CMP A  #$D       cohort
01410 E0A3 27 E0         BEQ    C1
01420 E0A5 81 5E         CMP A  #'^       UP ARROW?
01430 E0A7 20 2C         BRA    AL3       BRANCH FOR ADJUSTMENT
01440 E0A9 01            NOP

01460                    *INPUT HEX CHARACTER
01470 E0AA 8D CC   INHEX  BSR    INCH
01480 E0AC 80 30   INHEX1 SUB A  #$30
01490 E0AE 2B 4C         BMI    C3
01500 E0B0 81 09         CMP A  #$9
01510 E0B2 2F 0A         BLE    IN1HG
01520 E0B4 81 11         CMP A  #$11
01530 E0B6 2B 44         BMI    C3        NOT HEX
01540 E0B8 81 16         CMP A  #$16
01550 E0BA 2E 40         BGT    C3        NOT HEX
01560 E0BC 80 07         SUB A  #7
01570 E0BE 39      IN1HG RTS

01590 E0BF A6 00   OUT2H LDA A  0,X       OUTPUT 2 HEX CHAR
```



```
01600 E0C1 8D A4   OUT2HA BSR   OUTHL    OUT LEFT HEX CHAR
01610 E0C3 A6 00         LDA A  0,X
01620 E0C5 08            INX
01630 E0C6 20 A3         BRA    OUTHR     OUTPUT RIGHT HEX CHAR

01650 E0C8 8D F5   OUT4HS BSR   OUT2H     OUTPUT 4 HEX CHAR + SPACE
01660 E0CA 8D F3   OUT2HS BSR   OUT2H     OUTPUT 2 HEX CHAR + SPACE

01680 E0CC 86 20   OUTS   LDA A #$20      SPACE
01690 E0CE 20 A5         BRA    OUTCH     (BSR & RTS)


01710                    *ENTER POWER ON SEQUENCE
01720 E0D0 8E A042 START  LDS    #STACK
01730 E0D3 20 2C         BRA    AL1       BRANCH FOR ADDRESS COMPATIBIL

01750                    ********************************************
01760                    *PART OF MEMORY EXAMINE AND CHANGE
01770 E0D5 26 07   AL3    BNE    SK1
01780 E0D7 09            DEX
01790 E0D8 09            DEX
01800 E0D9 FF A00D       STX    XHI
01810 E0DC 20 AC         BRA    CHA51
01820 E0DE FF A00D SK1   STX    XHI
01830 E0E1 20 02         BRA    AL4

01850 E0E3 20 6D   E0E3   BRA    CONTRL    BRANCH FOR MIKBUG EQUIV. CONT

01870 E0E5 81 30   AL4    CMP A  #$30
01880 E0E7 25 A1         BCS    CHA51
01890 E0E9 81 46         CMP A  #$46
01900 E0EB 22 9D         BHI    CHA51
01910 E0ED 8D BD         BSR    INHEX1
01920 E0EF BD E057       JSR    BYTE1
01930 E0F2 09            DEX
01940 E0F3 A7 00         STA A  0,X       CHANGE MEMORY
01950 E0F5 A1 00         CMP A  0,X
01960 E0F7 27 91         BEQ    CHA51     DID CHANGE
01970 E0F9 7E E040       JMP    LOAD19    DIDN'T CHANGE
01980 E0FC BE A008 C3    LDS    SP
01990 E0FF 20 49         BRA    SWTCTL
02000                    ************************************************

02020                    *CONTINUE POWER UP SEQUENCE
02030 E101 BF A008 AL1   STS    SP        INIT TARGET STACK PTR.
02040 E104 86 FF         LDA A  #$FF
02050 E106 BD E308       JSR    SWISET
02060                    *CONFIGURE FOR PIA AND SEE IF OK
02070 E109 CE 8004       LDX    #CTLPOR
02080 E10C BD E284       JSR    PIAINI    INIT PIA
02090 E10F A6 00         LDA A  0,X
02100 E111 A1 02         CMP A  2,X
02110 E113 20 02         BRA    AL2
```

```
02130 E115 20 19   E115   BRA    PRINT    BRA FOR BILOAD

02150 E117 26 39   AL2    BNE    CONTRL

02170              *INITIALIZE AS ACIA
02180 E119 86 03          LDA A  #3       ACIA MASTER RESET
02190 E11B A7 00          STA A  0,X
02200 E11D 86 11          LDA A  #$11
02210 E11F A7 00          STA A  0,X
02220 E121 20 2F          BRA    CONTRL

02240              *ENTER FROM SOFTWARE INTERRUPT
02250 E123 01      SFO    NOP
02260 E124 BF A008 SFE1   STS    SP       SAVE TARGETS STACK POINTER
02270              *DECREMENT P COUNTER
02280 E127 30             TSX
02290 E128 6D 06          TST    6,X
02300 E12A 26 02          BNE    *+4
02310 E12C 6A 05          DEC    5,X
02320 E12E 6A 06          DEC    6,X
02330              *PRINT CONTENTS OF STACK
02340 E130 CE E19D PRINT  LDX    #MCL
02350 E133 BD E07E        JSR    PDATA1
02360 E136 FE A008        LDX    SP
02370 E139 08             INX
02380 E13A 8D 8E          BSR    OUT2HS   COND CODES
02390 E13C 8D 8C          BSR    OUT2HS   ACC B
02400 E13E 8D 8A          BSR    OUT2HS   ACC A
02410 E140 8D 86          BSR    OUT4HS   IXR
02420 E142 8D 84          BSR    OUT4HS   PGM COUNTER
02430 E144 CE A008        LDX    #SP
02440 E147 BD E0C8        JSR    OUT4HS   STACK POINTER
02450 E14A FE A012 SWTCTL LDX    SWIJMP
02460 E14D 8C E123        CPX    #SFO
02470 E150 27 19          BEQ    CONTR1

02490 E152 8E A042 CONTRL LDS    #STACK   SET CONTRL STACK POINTER
02500 E155 CE 8004        LDX    #CTLPOR  RESET TO CONTROL PORT
02510 E158 FF A00A        STX    PORADD
02520 E15B 7F A00C        CLR    PORECH   TURN ECHO ON
02530 E15E 8D 73          BSR    SAVGET   GET PORT # AND TYPE
02540 E160 27 03          BEQ    POF1
02550 E162 BD E27D        JSR    PIAECH   SET PIA ECHO ON IF MP-C INTER
02560 E165 BD E353 POF1   JSR    PNCHOF   TURN PUNCH OFF
02570 E168 BD E347        JSR    RDOFF    TURN READER OFF
02580 E16B CE E19C CONTR1 LDX    #MCLOFF
02590 E16E BD E07E        JSR    PDATA1   PRINT DATA STRING
02600 E171 8D 39          BSR    INEEE    READ COMMAND CHARACTER

02620              *COMMAND LOOKUP ROUTINE
02630 E173 CE E3D1 LOOK   LDX    #TABLE
02640 E176 A1 00   OVER   CMP A  0,X
02650 E178 26 07          BNE    SK3
02660 E17A BD E0CC        JSR    OUTS     SKIP SPACE
```



```
02670 E17D EE 01          LDX    1,X
02680 E17F 6E 00          JMP    0,X
02690 E181 08      SK3    INX
02700 E182 08             INX
02710 E183 08             INX
02720 E184 8C E3F8        CPX    #TABEND+3
02730 E187 26 ED          BNE    OVER
02740 E189 20 BF   SWTL1  BRA    SWTCTL

02760              *SOFTWARE INTERRUPT ENTRY POINT
02770 E18B FE A012 SFE    LDX    SWIJMP   JUMP TO VECTORED SOFTWARE INT
02780 E18E 6E 00          JMP    0,X


02800 E190 53      S9     FCB    'S,'9,4  END OF TAPE
      E191 39
      E192 04

02820              ************************************************
02830 E193 0D      MTAPE1 FCB    $D,$A,$15,0,0,0,'S,'1,4  PUNCH FORMAT
      E194 0A
      E195 15
      E196 00
      E197 00
      E198 00
      E199 53
      E19A 31
      E19B 04

02850 E19C 13      MCLOFF FCB    $13      READER OFF
02860 E19D 0D      MCL    FCB    $D,$A,$15,0,0,0,'$,4
      E19E 0A
      E19F 15
      E1A0 00
      E1A1 00
      E1A2 00
      E1A3 24
      E1A4 04

02880 E1A5 20 4C   EIA5   BRA    BILD     BINARY LOADER INPUT
02890              ************************************************


02920              *NMI SEQUENCE
02930 E1A7 FE A006 NMIV   LDX    NMI      GET NMI VECTOR
02940 E1AA 6E 00          JMP    0,X

02960 E1AC 20 40   INEEE  BRA    INEEE1

02980              *BYTE SEARCH ROUTINE
02990 E1AE BD E047 SEARCH JSR    BADDR    GET TOP ADDRESS
03000 E1B1 FF A004        STX    ENDA
03010 E1B4 BD E047        JSR    BADDR    GET BOTTOM ADDRESS
03020 E1B7 BD E055        JSR    BYTE     GET BYTE TO SEARCH FOR
```

E800 - EFFF


```
03030 E1BA 16              TAB
03040 E1BB A6 00    OVE    LDA A  0,X
03050 E1BD FF A00D         STX    XHI
03060 E1C0 11              CBA
03070 E1C1 27 02           BEQ    PNT
03080 E1C3 20 21           BRA    INCR1
03090 E1C5 CE E19D  PNT    LDX    #MCL
03100 E1C8 BD E07E         JSR    PDATA1
03110 E1CB CE A00D         LDX    #XHI
03120 E1CE 20 10           BRA    SKP0
03130              ************************************************

03150              *GO TO USER PROGRAM ROUTINE
03160 E1D0 3B      GOTO   RTI
03170 E1D1 20 3A   OUTEEE BRA     OUTEE1



03210              *SAVE IXR AND LOAD IXR WITH CORRECT
03220              *PORT NUMBER AND TEST FOR TYPE
03230 E1D3 FF A010 SAVGET STX     XTEMP    STORE INDEX REGISTER
03240 E1D6 FE A00A GETPT1 LDX     PORADD
03250 E1D9 37      ISACIA PSH B
03260 E1DA E6 01           LDA B  1,X
03270 E1DC E1 03           CMP B  3,X
03280 E1DE 33              PUL B
03290 E1DF 39              RTS
03300              ************************************************

03320              *CONTINUATION OF SEARCH ROUTINE
03330 E1E0 BD E0C8 SKP0   JSR     OUT4HS
03340 E1E3 FE A00D         LDX    XHI
03350 E1E6 BC A004 INCR1   CPX    ENDA
03360 E1E9 27 9E           BEQ    SWTL1
03370 E1EB 08              INX
03380 E1EC 20 CD           BRA    OVE

03400 E1EE 8D 06   INEEE1 BSR     INCH8    INPUT 8 BIT CHARACTER
03410 E1F0 84 7F           AND A  #%01111111  GET RID OF PARITY BIT
03420 E1F2 39              RTS

03440 E1F3 31      BILD   INS               FIX UP STACK WHEN USING
03450 E1F4 31             INS               BINARY LOADER ON SWTPC TAPES
03460 E1F5 31             INS

03480              *INPUT ONE CHAR INTO ACC A
03490 E1F6 37      INCH8  PSH B             SAVE ACC B
03500 E1F7 8D DA          BSR    SAVGET    SAVE IXR, GET PORT# AND TYPE
03510 E1F9 26 28          BNE    IN1       INPUT FROM PIA IF NOT
03520 E1FB 86 15          LDA A  #$15      RECONFIG FOR 8 BIT, 1 SB
03530 E1FD A7 00          STA A  0,X
03540 E1FF A6 00   ACIAIN LDA A  0,X
03550 E201 47             ASR A
03560 E202 24 FB          BCC    ACIAIN    NOT READY
```



```
03570 E204 A6 01           LDA A  1,X       LOAD CHAR
03580 E206 F6 A00C         LDA B  PORECH
03590 E209 27 07           BEQ    ACIOUT    ECHO
03600 E20B 20 11           BRA    RES       DON'T ECHO

03620              *OUTPUT ONE CHARACTER
03630 E20D 37      OUTEE1 PSH B             SAVE ACC B
03640 E20E 8D C3           BSR    SAVGET
03650 E210 26 2E           BNE    IOUT

03670 E212 C6 11   ACIOUT LDA B  #$11
03680 E214 E7 00           STA B  0,X
03690 E216 E6 00   ACIOU1 LDA B  0,X
03700 E218 57              ASR B
03710 E219 57              ASR B
03720 E21A 24 FA           BCC    ACIOU1    ACIA NOT READY
03730 E21C A7 01           STA A  1,X       OUTPUT CHARACTER
03740 E21E 33      RES    PUL B             RESTORE ACC B
03750 E21F FE A010         LDX    XTEMP
03760 E222 39              RTS

03780              *PIA INPUT ROUTINE
03790 E223 A6 00   IN1    LDA A  0,X        LOOK FOR START BIT
03800 E225 2B FC          BMI    IN1
03810 E227 8D 3A          BSR    DDL       DELAY HALF BIT TIME
03820 E229 C6 04          LDA B  #4         SET DEL FOR FULL BIT TIME
03830 E22B E7 02          STA B  2,X
03840 E22D 58             ASL B             SET UP CNTR WITH 8
03850 E22E 8D 2A   IN3    BSR    DEL        WAIT ONE CHAR TIME
03860 E230 0D             SEC
03870 E231 69 00          ROL    0,X
03880 E233 46             ROR A
03890 E234 5A             DEC B
03900 E235 26 F7          BNE    IN3
03910 E237 8D 21          BSR    DEL        WAIT FOR STOP BIT
03920 E239 F6 A00C        LDA B  PORECH     IS ECHO DESIRED?
03930 E23C 27 13          BEQ    IOUT2      ECHO
03940 E23E 20 DE          BRA    RES        RESTORE IXR, ACCB
03950              *PIA OUTPUT ROUTINE
03960 E240 8D 23   IOUT   BSR    DDL1       DELAY ONE HALF BIT TIME
03970 E242 C6 0A          LDA B  #$A        SET UP COUNTER
03980 E244 6A 00          DEC    0,X        SET START BIT
03990 E246 8D 16          BSR    DE         START TIMER
04000 E248 8D 10   OUT1   BSR    DEL        DELAY ONE BIT TIME
04010 E24A A7 00          STA A  0,X        PUT OUT ONE DATA BIT
04020 E24C 0D             SEC
04030 E24D 46             ROR A             SHIFT IN NEXT BIT
04040 E24E 5A             DEC B             DECREMENT COUNTER
04050 E24F 26 F7          BNE    OUT1       TEST FOR 0
04060 E251 E6 02   IOUT2  LDA B  2,X        TEST FOR STOP BITS
04070 E253 58             ASL B             SHIFT BIT TO SIGN
04080 E254 2A C8          BPL    RES        BRA FOR 1 STOP BIT
04090 E256 8D 02          BSR    DEL        DELAY FOR STOP BITS
04100 E258 20 C4          BRA    RES
```

```
04110 E25A 6D 02    DEL     TST     2,X       IS TIME UP
04120 E25C 2A FC            BPL     DEL
04130 E25E 6C 02    DE      INC     2,X       RESET TIMER
04140 E260 6A 02            DEC     2,X
04150 E262 39               RTS

04170 E263 6F 02    DDL     CLR     2,X       HALF BIT DELAY
04180 E265 8D F7    DDL1    BSR     DE
04190 E267 20 F1            BRA     DEL


04220               *OPTIONAL PORT ROUTINE
04230 E269 8D 83    OPTL    BSR     INEEE1
04240 E26B 16               TAB
04250 E26C 7F A00B          CLR     PORADD+1  SET I/O ADDRESS FOR $8000
04260 E26F FE A00A          LDX     PORADD
04270 E272 8D 10            BSR     PIAINI    INITIALIZE PIA
04280 E274 8D 07            BSR     PIAECH    SET ECHO
04290 E276 CE E3EF          LDX     #TABLE1   P, L OR E
04300 E279 17               TBA
04310 E27A 7E E176          JMP     OVER      LOOK AT TABLE FOR E, L OR P

04330 E27D 86 34    PIAECH  LDA A   #$34      SET DDR
04340 E27F A7 03            STA A   3,X
04350 E281 A7 02            STA A   2,X
04360 E283 39       NOOPT   RTS

04380               *PIA INITIALIZATION ROUTINE
04390 E284 6C 00    PIAINI  INC     0,X       SET DDR
04400 E286 86 07            LDA A   #$7
04410 E288 A7 01            STA A   1,X
04420 E28A 6C 00            INC     0,X
04430 E28C A7 02            STA A   2,X
04440 E28E 39               RTS

04460               *MINIFLOPPY DISK BOOT
04470 E28F 7F 8014  DISK    CLR     $8014
04480 E292 8D 2E            BSR     DELAY
04490 E294 C6 0B            LDA B   #$0B
04500 E296 8D 25            BSR     RETT2
04510 E298 E6 04    LOOP1   LDA B   4,X
04520 E29A C5 01            BIT B   #1
04530 E29C 26 FA            BNE     LOOP1
04540 E29E 6F 06            CLR     6,X
04550 E2A0 8D 1D            BSR     RETURN
04560 E2A2 C6 9C            LDA B   #$9C
04570 E2A4 8D 17            BSR     RETT2
04580 E2A6 CE 2400          LDX     #$2400
04590 E2A9 C5 02    LOOP2   BIT B   #2
04600 E2AB 27 06            BEQ     LOOP3
04610 E2AD B6 801B          LDA A   $801B
04620 E2B0 A7 00            STA A   0,X
04630 E2B2 08               INX
04640 E2B3 F6 8018  LOOP3   LDA B   $8018
```



```
04650 E2B6 C5 01            BIT B   #1
04660 E2B8 26 EF            BNE     LOOP2
04670 E2BA 7E 2400          JMP     $2400
04680 E2BD E7 04    RETT2   STA B   4,X
04690 E2BF 8D 00    RETURN  BSR     RETT1
04700 E2C1 39       RETT1   RTS

04720               *GENERAL PURPOSE DELAY LOOP
04730 E2C2 CE FFFF  DELAY   LDX     #$FFFF
04740 E2C5 09       DELAY1  DEX
04750 E2C6 8C 8014          CPX     #$8014    STOP AT 8014
04760 E2C9 26 FA    DUM     BNE     DELAY1
04770 E2CB 39               RTS


04800               *CLRAR SCREEN FOR CT-1024 TYPE TERMINALS
04810 E2CC CE E009  CLEAR   LDX     #CURSOR
04820 E2CF BD E07E          JSR     PDATA1
04830 E2D2 8D F1            BSR     DELAY1    DELAY
04840 E2D4 BD E347  RDOFF1  JSR     RDOFF
04850 E2D7 20 58            BRA     C4


04870               *BREAKPOINT ENTERING ROUTNE
04880 E2D9 CE E123  BREAK   LDX     #SFO
04890 E2DC BC A012          CPX     SWIJMP    BREAKPOINTS ALREADY IN USE?
04900 E2DF 27 1A            BEQ     INUSE
04910 E2E1 08               INX
04920 E2E2 8D 32    BREAKO  BSR     STO1
04930 E2E4 BD E047          JSR     BADDR
04940 E2E7 FF A014          STX     BKPT
04950 E2EA A6 00            LDA A   0,X
04960 E2EC B7 A016          STA A   BKLST
04970 E2EF 86 3F            LDA A   #$3F
04980 E2F1 A7 00            STA A   0,X
04990 E2F3 CE E123          LDX     #SFO
05000 E2F6 8D 1E            BSR     STO1
05010 E2F8 7E E16B          JMP     CONTR1
05020 E2FB FE A014  INUSE   LDX     BKPT
05030 E2FE B6 A016          LDA A   BKLST
05040 E301 A7 00            STA A   0,X
05050 E303 CE E124          LDX     #SFE1
05060 E306 20 DA            BRA     BREAKO

05080 E308 B7 A043  SWISET  STA A   STACK+1   FIX POWER UP INTERRUPT
05090 E30B FE A012          LDX     SWIJMP
05100 E30E 8C E123          CPX     #SFO
05110 E311 27 06            BEQ     STORTN
05120 E313 CE E124  STO     LDX     #SFE1
05130 E316 FF A012  STO1    STX     SWIJMP
05140 E319 39       STORTN  RTS

05160 E31A 8D 5A    PUNCH1  BSR     PUNCH
05170 E31C 20 0F            BRA     POFC4
```

```
05190                  *FORMAT END OF TAPE WITH PGM. CTR. AND S9
05200 E31E CE A049 PNCHS9 LDX    #$A049
05210 E321 FF A004        STX    ENDA
05220 E324 09             DEX
05230 E325 8D 52          BSR    PUNCH2
05240 E327 CE E190        LDX    #S9
05250 E32A BD E07E PDAT   JSR    PDATA1
05260 E32D 8D 24   POFC4  BSR    PNCHOF
05270 E32F 8D 91          BSR    DELAY
05280 E331 7E E152 C4     JMP    CONTRL

05300 E334 73 A00C RDON   COM    PORECH   DISABLE ECHO FOR ACIA
05310 E337 86 11          LDA A  #$11     RON CHAR,
05320 E339 C6 20          LDA B  #$20     STROBE CHAR
05330 E33B 8D 1A          BSR    STROBE
05340 E33D BD E1D9        JSR    ISACIA   CHECK TO SEE IF PIA
05350 E340 27 04          BEQ    RTNN
05360 E342 86 3C          LDA A  #$3C     DISABLE PIA ECHO IF PIA
05370 E344 A7 03          STA A  3,X
05380 E346 39       RTNN  RTS

05400 E347 86 13  RDOFF   LDA A  #$13     TURN READER OFF
05410 E349 C6 10          LDA B  #$10
05420 E34B 20 0A          BRA    STROBE

05440 E34D 86 12  PNCHON  LDA A  #$12
05450 E34F C6 04          LDA B  #4
05460 E351 20 04          BRA    STROBE

05480 E353 86 14  PNCHOF  LDA A  #$14
05490 E355 C6 08          LDA B  #$8

05510                  *PIA STROBING ROUTINE FOR PUNCH/READ ON/OFF
05520 E357 BD E075 STROBE JSR    OUTCH
05530 E35A BD E1D6        JSR    GETPT1
05540 E35D 27 16          BEQ    RTN1
05550 E35F 86 02          LDA A  #2
05560 E361 CA 01          ORA B  #1
05570 E363 8D 0C          BSR    STR2
05580 E365 8D 08          BSR    STR1
05590 E367 86 02          LDA A  #2
05600 E369 C6 01          LDA B  #1
05610 E36B E7 00          STA B  0,X
05620 E36D 8D 02          BSR    STR2
05630 E36F 86 06  STR1    LDA A  #6
05640 E371 A7 01  STR2    STA A  1,X
05650 E373 E7 00          STA B  0,X
05660 E375 39       RTN1  RTS

05680                  *PUNCH FROM BEGINNING ADDRESS (BEGA) THRU
05690                  *ENDING ADDRESS (ENDA)
05700 E376 FE A002 PUNCH  LDX    BEGA
05710 E379 FF A044 PUNCH2 STX    TW
05720 E37C 8D CF          BSR    PNCHON
```



```
05730 E37E B6 A005 PUN11  LDA A  ENDA+1
05740 E381 B0 A045        SUB A  TW+1
05750 E384 F6 A004        LDA B  ENDA
05760 E387 F2 A044        SBC B  TW
05770 E38A 26 04          BNE    PUN22
05780 E38C 81 10          CMP A  #16
05790 E38E 25 02          BCS    PUN23
05800 E390 86 0F  PUN22   LDA A  #15
05810 E392 8B 04  PUN23   ADD A  #4
05820 E394 B7 A047        STA A  BYTECT
05830 E397 80 03          SUB A  #3
05840 E399 B7 A046        STA A  TEMP
05850                  *PUNCH C/R L/F NULLS S1
05860 E39C CE E193        LDX    #MTAPE1
05870 E39F BD E07E        JSR    PDATA1
05880 E3A2 5F             CLR B
05890                  *PUNCH FRAME COUNT
05900 E3A3 CE A047        LDX    #BYTECT
05910 E3A6 8D 24          BSR    PUNT2    PUNCH 2 HEX CHARACTERS
05920                  *PUNCH ADDRESS
05930 E3A8 CE A044        LDX    #TW
05940 E3AB 8D 1F          BSR    PUNT2
05950 E3AD 8D 1D          BSR    PUNT2
05960                  *PUNCH DATA
05970 E3AF FE A044        LDX    TW
05980 E3B2 8D 18  PUN32   BSR    PUNT2    PUNCH ONE BYTE
05990 E3B4 7A A046        DEC    TEMP
06000 E3B7 26 F9          BNE    PUN32
06010 E3B9 FF A044        STX    TW
06020 E3BC 53             COM B
06030 E3BD 37             PSH B
06040 E3BE 30             TSX
06050 E3BF 8D 0B          BSR    PUNT2    PUNCH CHECKSUM
06060 E3C1 33             PUL B           RESTORE STACK
06070 E3C2 FE A044        LDX    TW
06080 E3C5 09             DEX
06090 E3C6 BC A004        CPX    ENDA
06100 E3C9 26 B3          BNE    PUN11
06110 E3CB 39       RTN5  RTS

06130                  *PUNCH 2 HEX CHAR, UPDATE CHECKSUM
06140 E3CC EB 00  PUNT2   ADD B  0,X
06150 E3CE 7E E0BF        JMP    OUT2H    OUTPUT 2 HEX CHAR AND RTS

06170                  *COMMAND TABLE
06180 E3D1 47       TABLE FCB    'G       GOTO
06190 E3D2 E1D0           FDB    GOTO
06200 E3D4 5A             FCB    'Z       GOTO PROM
06210 E3D5 C000           FDB    PROM
06220 E3D7 4D             FCB    'M       MEMORY EXAM AND CHANGE
06230 E3D8 E088           FDB    CHANGE
06240 E3DA 46             FCB    'F       BYTE SEARCH
06250 E3DB E1AE           FDB    SEARCH
06260 E3DD 52             FCB    'R       REGISTER DUMP
```

```
06270 E3DE E130          FDB   PRINT
06280 E3E0 4A            FCB   'J       JUMP
06290 E3E1 E005          FDB   JUMP
06300 E3E3 43            FCB   'C       CLEAR SCREEN
06310 E3E4 E2CC          FDB   CLEAR
06320 E3E6 44            FCB   'D       DISK BOOT
06330 E3E7 E28F          FDB   DISK
06340 E3E9 42            FCB   'B       BREAKPOINT
06350 E3EA E2D9          FDB   BREAK
06360 E3EC 4F            FCB   'O       OPTIONAL PORT
06370 E3ED E269          FDB   OPTL
06380 E3EF 50    TABLE1  FCB   'P       ASCII PUNCH
06390 E3F0 E31A          FDB   PUNCH1
06400 E3F2 4C            FCB   'L       ASCII LOAD
06410 E3F3 E00C          FDB   LOAD
06420 E3F5 45    TABEND  FCB   'E       END OF TAPE
06430 E3F6 E31E          FDB   PNCHS9

06450 E3F8               ORG   $E3F8
06460 E3F8 E000          FDB   IRQV      IRQ VECTOR
06470 E3FA E18B          FDB   SFE       SOFTWARE INTERRUPT
06480 E3FC E1A7          FDB   NMIV      NMI VECTOR
06490 E3FE E0D0          FDB   START     RESTART VECTOR

06510 A048               ORG   $A048
06520 A048 E0D0          FDB   START
06530                    END

TOTAL ERRORS 00000
```



```
IRQ     A000
BEGA    A002
ENDA    A004
NMI     A006
SP      A008
PORADD  A00A
PORECH  A00C
XHI     A00D
XLOW    A00E
CKSM    A00F
XTEMP   A010
SWIJMP  A012
TW      A044
TEMP    A046
BYTECT  A047
CTLPOR  8004
PROM    C000
BKPT    A014
BKLST   A016
STACK   A042
IRQV    E000
JUMP    E005
CURSOR  E009
LOAD    E00C
LOAD3   E00F
LOAD11  E02B
LOAD15  E03B
LOAD19  E040
LOAD21  E044
BADDR   E047
BYTE    E055
BYTE1   E057
OUTHL   E067
OUTHR   E06B
OUTCH   E075
INCH    E078
PDATA2  E07B
PDATA1  E07E
C1      E085
CHANGE  E088
CHA51   E08A
ANOTH   E09B
INHEX   E0AA
INHEX1  E0AC
IN1HG   E0BE
OUT2H   E0BF
OUT2HA  E0C1
OUT4HS  E0C8
OUT2HS  E0CA
OUTS    E0CC
START   E0D0
AL3     E0D5
SK1     E0DE
E0E3    E0E3
AL4     E0E5
C3      E0FC
AL1     E101
E115    E115
AL2     E117
SFO     E123
SFE1    E124
PRINT   E130
SWTCTL  E14A
CONTRL  E152
POF1    E165
CONTR1  E16B
LOOK    E173
OVER    E176
SK3     E181
SWTL1   E189
SFE     E18B
S9      E190
MTAPE1  E193
MCLOFF  E19C
MCL     E19D
EIA5    E1A5
NMIV    E1A7
INEEE   E1AC
SEARCH  E1AE
OVE     E1BB
PNT     E1C5
GOTO    E1D0
OUTEEE  E1D1
SAVGET  E1D3
GETPT1  E1D6
ISACIA  E1D9
SKPO    E1E0
INCR1   E1E6
INEEE1  E1EE
BILD    E1F3
INCH8   E1F6
ACIAIN  E1FF
OUTEE1  E20D
ACIOUT  E212
ACIOU1  E216
RES     E21E
IN1     E223
IN3     E22E
IOUT    E240
OUT1    E248
IOUT2   E251
DEL     E25A
DE      E25E
DDL     E263
DDL1    E265
OPTL    E269
PIAECH  E27D
NOOPT   E283
PIAINI  E284
DISK    E28F
LOOP1   E298
LOOP2   E2A9
LOOP3   E2B3
RETT2   E2BD
RETURN  E2BF
RETT1   E2C1
DELAY   E2C2
DELAY1  E2C5
DUM     E2C9
CLEAR   E2CC
RDOFF1  E2D4
BREAK   E2D9
BREAKO  E2E2
INUSE   E2FB
SWISET  E308
STO     E313
STO1    E316
STORTN  E319
PUNCH1  E31A
PNCHS9  E31E
PDAT    E32A
POFC4   E32D
C4      E331
RDON    E334
RTNN    E346
RDOFF   E347
PNCHON  E34D
PNCHOF  E353
STROBE  E357
STR1    E36F
STR2    E371
RTN1    E375
PUNCH   E376
PUNCH2  E379
PUN11   E37E
PUN22   E390
PUN23   E392
PUN32   E3B2
RTN5    E3CB
PUNT2   E3CC
TABLE   E3D1
TABLE1  E3EF
TABEND  E3F5
```

T = spring naar CDAT

Z = laad BOS